AN
INTERNSHIP REPORT

ON

A FILE SYSTEM FOR MOBILE COMPUTING PROJECT

BY

KAMAL ACHARYA

(Tribhuvan University)

Date: 2022/01/10

Abstract

The recent proliferation of portable computers, the advent of personal digital assistants (PDAs) and continuing advances in computer networking all point to a future in which mobility of both users and computing system elements will be standard. However, these developments also invalidate many of the assumptions made by current system software, which has been built for stationary systems and users. An important component of system software that needs to looked at afresh is the file system. A user should be able to access the files he or she needs regardless of location. Although existing research efforts do provide some support for this, there are significant deficiencies that need to be remedied. We propose two related designs to address these problems. The first exploits the unique property of a PDA that it can accompany its owner at all times by using it to carry files of most immediate need to him or her. The second makes use of the recently proposed idea of a *computing persona*.

Table of Contents

1. Introduction

We are about to enter a new era in computing: the era of mobile computing. Computer hardware technology has progressed so rapidly that today's portable computers are more powerful than the desktop computers built just a few years ago. Computer networks have also been expanding at an astonishing pace, and there have been constant advancements in networking technologies. While most computer networks until now have been of the wired type, there is much promise of wireless networks becoming widely available in the near future. All these developments will provide the hardware base needed for mobile computing.

However, these advances in hardware cannot, by themselves, make mobile computing a common reality. Many of the assumptions upon which current system software is based will no longer be valid. As pointed out in [Satyanarayanan 1993b], there are relative differences between mobile and stationary computers that will always exist and that are not the result of shortcomings of current technology:

• *Mobile computers are resource-poor compared to stationary computers*. Due to constraints on weight, size and power consumption, mobile computers will always be inferior to stationary computers in multiple respects such as processing power and storage capacity. Moreover, there is also a strong possibility that technology advances for mobile computers will focus on reducing size and increasing battery life and, thus, the gap between them and fixed systems is likely to grow [Weiser 1993b]. In some cases, incremental advances in technology will be of no avail due to human limitations. For example, no matter how much resolution is improved, there will be a limit to what the human eye can see clearly on a tiny screen.

• *Mobile computers are more prone to loss, damage and theft than stationary computers*. Being moved around makes mobile computers more susceptible to loss or damage. And, of course, the street is not as safe as the office or home.

• *Mobile computers must operate under a much wider range of networking conditions than stationary ones*. Stationary computers are generally connected to a wired network that has reliable and well-defined characteristics. A mobile one on the other hand must make do with whatever wired or wireless connectivity is available at its current location, which might often be none at all.

These differences are very significant and will require a complete redesign of system software.

It is also likely that we will soon a see much greater variety of mobile computing devices in terms of functionality. Currently, most mobile computers are *portable workstations* i.e. machines that are lighter and smaller than desktop workstations but functionally very similar to them. They have full-size keyboards and screens that are typically 10" across; they can be used quite easily for tasks such as typing in a document. Recently however, we have seen the introduction of a new class of mobile computers popularly called *personal digital assistants (PDAs)*. Examples are the Apple Newton, the AT&T Eo and the IBM-BellSouth Simon. These PDAs are pocket-sized, have pads that serve as both screens and input devices, and are likely to be used quite differently from conventional computers. The very different character of these devices must be considered when designing system software for them.

Mobility of computers is not the only kind of mobility that will have to be dealt with. Even in cases where elements of a distributed system are stationary, its users are mobile, and there are numerous problems to be solved. The movement of a user from one location in the system to another should in general make as little difference as possible to the way that user interacts with the system. For example, the user would like to be able to access the same files and applications and possibly even have running applications move as he or she moves. Mail intended for the user should always be delivered to his or her current location.

There are, of course, some things that should not be location-independent. As a simple example, the user would presumably want the clock display to show the local time. Or, if a user moves from a terminal that can display only text to one that can display graphics, applications should take advantage of this additional capability. We thus see the need for system software to insulate the user from most changes caused by mobility while adapting to or even exploiting others.

Each user has a certain set of preferences as to the environment provided by the computing system. In a fairly static system, a user profile stored in a file would suffice to define these preferences. However, under the widely varying conditions introduced by mobility, this will probably be insufficient. In [Banerji 1993], the concept of a *computing persona* has been proposed to deal with this problem. Associated with each user is a customized persona that represents the computing environment

that is closest to the user's preferences under the prevailing conditions. When the user moves to a new location, the persona is re-established there with as little modification as is possible. When changes in physical conditions so demand, the user's persona will change according to specified policies.

An important component of system software that mobile computing forces us to take a fresh look at is the file system. An isolated computer has its own file system while one that is part of an interconnection can use a shared file system in which files may be stored on remote computers. Existing file systems assume that a computer falls cleanly into one of these two categories. However, a mobile computer can be an isolated computer at times and part of a distributed system at others. It can also move from one distributed system to another or be connected to a single and otherwise isolated computer through a short-range link. How does one design a file system for such a computer so that its user can effectively operate in these different situations and move from one to another in a seamless manner? This problem will form the focus of this document.

In the next chapter, we take a broad look at work that has been done in the area of mobile computing. In Chapter 3, we review existing distributed file systems that provide support for mobile computing. In the following chapter, we point out where these systems fall short and present a design that addresses these problems. The final chapter presents some early ideas for implementation of the system.

2. Mobile Computing

This chapter provides an overview of the different directions being taken to support mobile computing. It will first cover an approach to future computing that is extensively based on the use of mobile computers. Next, we take a look at work that has been done in the area of location-independent computing. Finally, we touch on work being done to support networking of mobile computers.

2.1 Ubiquitous Computing

An important research effort that is intimately connected with mobile computing is the ubiquitous computing (Ubicomp) project at Xerox PARC. [Weiser 1991] introduced a vision of the future of computing in which scores of computers of widely varying sizes and kinds are scattered inconspicuously throughout the physical environment and their use by people becomes unconscious. This vision is based on the idea that it is only when computers fade into the background, instead of being the focus of attention, that they will become truly effective tools in our everyday life.

As the first step in the project, Weiser and his colleagues have built and deployed in their offices computing devices of three different sizes. The first is a wall-mounted interactive surface called a *board* that is roughly speaking an intelligent whiteboard. The second is a notebook-sized device called a *pad* which has a writing and display surface. The third and smallest kind of device, called a *tab*, is palm-sized and consists of a monochrome display with a transparent pressure-sensitive screen on top and buttons on the side. A typical room will contain hundreds of tabs, tens of pads and one or two boards.

Both the pads and the tabs are mobile devices and clearly most of the communication will have to be wireless. Ubiquitous computing and mobile computing are therefore very closely tied together if not two terms for the same thing. In [Weiser 1993a], several of the computer science issues that have to be addressed in order to make ubiquitous computing a reality are raised. New hardware will have to be developed e.g. low-power devices, wireless networks capable of handling hundreds of devices within a small space, and pen-based devices. New network protocols are needed for wireless media access. New substrates for human-computer interaction have also to be developed. For exam-

ple, tabs are too small for a keyboard, so an alternative means of input has to be used. Ubiquitous computing will require a whole new set of applications e.g. ones for locating people and shared drawing. Mechanisms have to be created that will allow individuals control over the accumulation and dissemination of information about them such as their location [Spreitzer 1993].

Ubicomp is already been applied to a real problem: that of providing responsive office environments that automatically control conditions such as temperature and lighting condition in a manner that satisfies the occupants while conserving energy [Elrod 1993]. Several offices within Xerox PARC have been outfitted with temperature, light-level, occupancy, and active badge [Want 1992] sensors together with computer-controlled ventilation, heating, electrical outlets, and overhead lighting. These devices are connected to a conventional computerized building management system. Occupants of an office can control the lighting and temperature using a tab. By automatically shutting off lights and lowering the heating or cooling of rooms when they are unoccupied, energy savings can be realized.

2.2 Location-independent Computing

In general, users want their mobility to make as little difference as possible to the way they interact with their computing system. For this reason, location independence is already an important issue in current distributed systems and is likely to become even more important once mobile computing devices become a common part of such systems. Some distributed file systems such as the Andrew File System [Howard 1988] provide location-independence by using a global name space. Others like the Prospero File System [Neuman 1992] provide a location-independent name space to each user but allow different users to have different name spaces. The Prospero Directory Service attempts to extend this user-centered location-independent view to computing resources other than just files [Neuman 1993].

With Prospero, each user defines a *virtual system* which is a hierarchical name space for the available computing resources. Since a virtual system is customized for a particular user, different users see different name spaces. However, as a user moves from one location to another, he or she sees the same name space. Also, the binding of names to objects does not have to be static. Rather, a name can be used to refer to certain logical attributes so that as changes occur (such as in the location

of the user) the binding changes to the new object that satisfies those attributes.

An example user virtual system is shown in Figure 1. The ROOT link refers to the root of the user's virtual file system, which is a Prospero file system that provides the same representation regardless of location. The SESSIONS link points to the collection of sessions the user has currently established. An entry is automatically added to the SESSIONS directory each time the user logs in. This entry will point to the virtual system of the platform on which the user logged in, which will contain various platform-specific information.

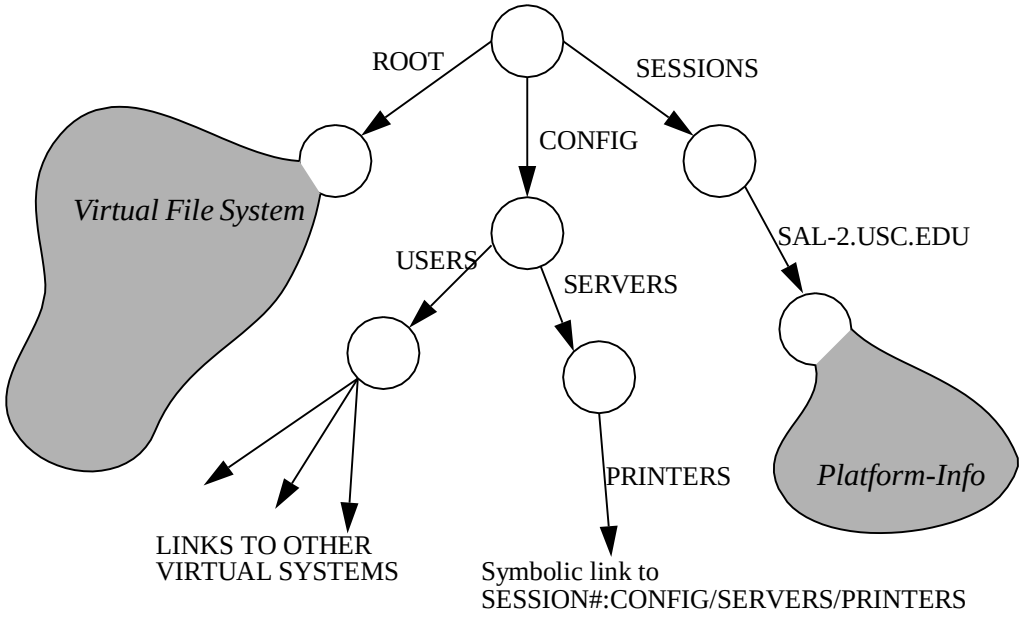


Figure 1: An Example Prospero Virtual System

The servers that the user has selected are included in the CONFIG/SERVERS directory. For example, the CONFIG/SERVERS/PRINTERS directory defines the available printers. These printers do not have to be explicitly defined. Instead a symbolic link to a directory of printers for the session currently being used can be defined using a *virtual system alias*. Such an alias ends with the string #: e.g. SESSION#:CONFIG/SERVERS/PRINTERS, which will be resolved by taking the sub-string following the #: and resolving it in the virtual system of the user's current session.

Prospero also provides two useful features called *union links* and *filters*. When a union link is added to a directory, it results in a directory that is the union of the original directory and the directory pointed at by the union link. For example, a user could obtain a combined view of the printers at home and at office by creating union links to the individual directories representing these collections.

Filters are functions applied to a directory to select only a portion of the objects in it based on certain criteria. For example, an **attribute()** filter can be applied to a directory of printers to select only those printers that have the attribute POSTSCRIPT.

2.3 Networking

Much work needs to be done in the area of networking for mobile computers since the field of wireless networks is still in its infancy. Most of the current network protocols assume that all computers are stationary. Also, a fairly large and constant bandwidth (10 Mb/s for Ethernet) is assumed. Furthermore, while there are capital costs incurred in setting up a network, operational costs are low and so network communication is assumed to be essentially free. None of these assumptions will continue to hold in the case of the wireless networks that will commonly be used by mobile computers. New network architectures and protocols must therefore be devised to deal with these changed conditions.

One approach to dealing with mobility that seems to be gaining popularity is to have a *proxy* [Watson 1993] or *agent* [Adams 1993] [Athan 1993] for each mobile computer. This proxy resides at a fixed network address and is responsible for keeping track of the whereabouts of the mobile host. The mobile host is known to other hosts on the network only by its proxy's address and so all messages intended for it are sent to its proxy which then does the necessary forwarding.

In [Bhagwat 1993], a scheme has been presented that uses an existing option in the Internet Protocol (IP) to provide transparent network access to mobile hosts. This option called the Loose Source Route (LSR) option provides a means for the source of a datagram to provide partial routing information that can be used by routers to forward it to the destination. Return traffic to the originator of an LSR packet is also sent with the LSR option by reversing the route taken by the incoming packets. While the initial packets may have to travel by a sub-optimal route (because the current location of a mobile host is not known to the source), subsequent packets use the optimal route. It should be noted, however, that there are potential security problems associated with use of the LSR option.

The system is organized in the following manner. Multiple subnets are reserved for mobile hosts. These subnets are logical and not physical. Each such subnet has a mobile router (MR) which keeps track of the current physical location of the MHs assigned to that subnet and routes packets destined for them. A mobile host (MH) must physically lie within the cell of a Mobile Access Station

(MAS) to communicate with hosts on the wired network. The MASs act as gateways between the wired and wireless parts of the network.

When an MH initiates communication with a stationary host (SH), the packet goes to the MAS, which sends it to the SH by the optimal route since the SH's location is fixed and well-known. The reply packets from the SH to the MH use the LSR option to reverse the route and so the optimal route is used in the reverse direction too. On the other hand, if an SH initiates communication with an MH, it will be unaware of the MH's current location. Therefore, the initial packets will go to the MR responsible for that MH, which will forward it to the MH's current MAS, which will in turn deliver it to the MH. Reply packets from the MH to the SH, however will be sent by the optimal route by the MAS using the LSR option. Once the SH receives such a reply packet, it sends subsequent packets to the MH along the optimal route by reversing the LSR option.

When an MH (say MH1) initiates communication with another MH (say MH2), its MAS checks to see if MH2 is in the same cell. If so, the packets are sent directly to MH2 and reply packets use the reverse route back to MH1. (If the wireless technology supports direct MH to MH communication, MH1 and MH2 can bypass the MAS once they learn they are in the same cell.) However, if MH2 was in different cell from MH1, the initial packets would be sent by MH1's MAS (MAS1) to the MR for MH2, which would send it to the MH2's MAS (MAS2). Reply packets would be sent by MAS2 by the optimal route to MAS1 and once such a packet is received by MAS1, it can send subsequent packets destined for MH2 directly to MAS2.

3. Distributed File Systems

Most work in the area of file systems for mobile computing has focussed on providing support for mobile computers within distributed file systems. This chapter will cover three such efforts: the Coda File System, the LITTLE WORK project's support for disconnected operation for AFS, and the Ficus distributed file system.

3.1 The Coda File System

The Coda File System [Satyanarayanan 1990], developed at Carnegie Mellon University, is a distributed file system that provides support for portable computers. Coda bears a strong resemblance to its ancestor the Andrew File System (AFS) [Howard 1988] in many ways. Like AFS, it is designed for an environment consisting of a large number of untrusted Unix clients and a much smaller number of trusted Unix file servers. The servers collectively provide a single location-independent tree-structured name space to clients. This name space is divided into sub-trees called *volumes* and storage of files at individual servers takes place at this granularity. Each client has a cache manager named Venus that caches files from the servers on its local disk in order to provide reasonable performance.

AFS has been largely successful in meeting its design goals of providing secure, shared file access in a distributed system while scaling gracefully. Furthermore, while it was not designed to support mobile computers, its provision of a location-independent name space was an important step towards support for mobile users. However, a major problem with it is that failures of servers or the network severely compromise availability of data. Coda seeks to address this problem through two mechanisms: *server replication* and *disconnected operation* [Kistler 92]. Server replication is a feature whereby a single volume has read-write replicas at more than one server. The set of replication sites for a volume is called its *volume storage group (VSG*). The subset of the VSG currently accessible at a client is that client's *accessible VSG (AVSG)*. As long as there is at least one server in a client's AVSG, all files in that volume are available at that client for both reading and writing.

Disconnected operation occurs when the AVSG becomes empty. This may be caused by server or network failure (an involuntary disconnection) or the unplugging of a portable computer

from the network (a voluntary disconnection). In either case, the client must rely on the current contents of its cache to service file system requests. Coda supports disconnected operation by pre-caching files the user is most likely to need while disconnected, allowing all operations on these files during disconnection and then reintegrating changes upon reconnection. Support for mobile computers has been treated as a special case of the general problem of supporting disconnected operation.

Both server replication and disconnected operation are based on an *optimistic* replication strategy that allows updates in all partitions of a distributed system, with conflicts being detected and resolved after healing of partitions. The success of such a strategy depends on the actual occurrence of conflicts being relatively rare. Since concurrent write sharing is generally infrequent in Unix environments [Baker 1991] [Ousterhout 1985], such an assumption is in fact reasonable to make.

It is Coda's support for disconnected operation that is most interesting from the point of view of mobile computing and so it will be instructive to see in some detail how this support is provided. Most of the burden of providing this support is placed on Venus, the cache manager at each client. Venus rotates between three states: hoarding, emulation and reintegration. Hoarding is the normal state - here the client is connected and Venus is constantly hoarding files in its cache in preparation for possible disconnection. When disconnection actually occurs, Venus moves into the emulation state, so named because it must emulate certain server functions. Upon reconnection, it moves into the reintegration state, in which it resynchronizes its cache with the AVSG. When this has completed, it goes back into the hoarding state.

3.1.1 Hoarding

Traditionally, caching is used for reasons of performance. The objective of the cache manager is therefore to predict the data references that will be made in the immediate future and prefetch such data into the cache. In the case of Coda, the cache is being used for a second purpose - as a store for data that may not be referenced in the near future but whose availability is still critical. Therefore, while hoarding, Venus must balance these two possibly conflicting objectives while deciding which files to include in the cache. It does this by considering both objectives while assigning priorities to files and retaining the files with the highest priorities.

Venus uses both implicit information and explicit hints from the user while assigning priori-

ties. The implicit information is the recent file reference history that is normally used for cache management. The explicit hints are contained in a *hoard database (HDB)* residing on the client that indicates which file objects (files and directories) are critical to the user during disconnection and what their relative priorities, called *hoard priorities*, are. Entries in the HDB consist of pathnames of file objects along with their hoard priorities. The overall priority of a file object is a function of both its hoard priority and a measure that is based on its recent usage.

The user modifies the HDB using command scripts called *hoard profiles*. A hoard profile specifies objects to be added and deleted from the HDB and their hoard priorities. There is also support for meta-expansion of HDB entries i.e. one entry can specify an object together with all its children or all its descendants.

The cache is said to be in equilibrium when no uncached object has a higher priority than a cached object. Routine file usage as well as modifications to the HDB result in changes to the priorities of objects, thereby upsetting equilibrium. Venus performs periodic *hoard walks* to restore equilibrium. A hoard walk consists of first recomputing meta-expansions of HDB entries, then reevaluating priorities of all objects in the cache and finally replacing objects as necessary to restore equilibrium.

3.1.2 Emulation

Disconnection causes Venus to move into the emulation state. Since servers are inaccessible, Venus has to perform many of the functions usually performed by them, such as access checks. It also generates temporary file identifiers (fids) for new objects, which are converted to permanent ones during reintegration. Cache entries of deleted objects are freed while those of updated objects are given infinite priority so that they are not purged before reintegration. The changes made to the local cache must eventually make their way to the server upon reconnection. Venus must therefore keep a record of all mutating operations in sufficient detail that, later on, servers can validate these changes and then apply them to the server copies. Therefore, a log of all mutating operations, called a *replay log*, is maintained for each volume.

However it is also important that Venus make the most efficient use of the limited local disk space while doing this, so several optimizations are used to keep the log small. One such optimization is to record the `open`, `close` and intervening `write` operations on a single file as a single `store`

operation that occurs at the time of the `close`. Also, when a new `store` record is appended to the log, all previous `store` records pertaining to that file can be discarded. Further, when one operation merely serves to nullify the effect of a previous operation, the records for both operations can be deleted from the log.

The shutdown of a disconnected client should not result in loss of data and a crash should not result in more loss of data than if the machine were connected. Therefore, the cache contents as well as any meta-data must be kept in non-volatile storage. Venus uses a transaction facility called Recoverable Virtual Memory (RVM) [Satyanarayanan 1993a] to provide persistent storage as well as a recovery mechanism for its meta-data. The meta-data is periodically flushed to disk with Venus able to schedule the flushes. During emulation, Venus plays safe by performing these flushes more frequently.

3.1.3 Reintegration

Upon reconnection with one or more servers, Venus must propagate to the servers the changes made locally while disconnected, as well as refresh its cache to reflect updates that have been made to the servers by other clients in the system. Reintegration is performed a volume at a time, with update activity to the volume being suspended during this time.

Venus first replaces any temporary fids used in the replay log with permanent fids. It then ships the replay log in parallel to the AVSG. Each server performs the replay within a single transaction as follows. First, the transaction is begun, the log is parsed and all objects referenced by it are locked. Next, each operation is checked for conflict detection, protection, etc., and is executed if valid. In the case of `store` operations, everything except for actual data transfer is done in this step. The data transfers for all such operations are then performed in the next step. Finally, the transaction commits and all locks are released.

If reintegration succeeds, Venus frees the replay log and resets the priority of the cached objects it referenced. Otherwise, it writes the log out to a local replay file and purges the log and the cache entries of referenced objects. A tool is provided to allow the user to inspect the contents of the replay file, compare it to the state at the AVSG, and replay it selectively or entirely.

The only conflicts Coda has to detect are write/write conflicts, since the UNIX file system has no notion of atomicity beyond the boundary of a single system call. Each object is tagged with a *storeid* that uniquely identifies the last update to it. When replaying the log, the server has only to compare the storeid mentioned in the log with the one for its own replica to tell whether a conflict exists or not. If a conflict occurs on a directory, it is usually due to two *different* entries in the directory being updated concurrently; Coda can automatically resolve such conflicts by simply merging the two versions. However, in the case of files, resolution must be left to the user since Coda does not understand their semantic content.

3.2 Disconnected Operation for AFS

The LITTLE WORK project [Honeyman 1992] at the University of Michigan is an attempt to support portable computers within existing distributed computing environments. As part of this project, disconnected operation support for AFS has been developed through modifications to the cache manager at the AFS client [Huston 1993]. This work is quite similar to Coda in that it uses the client's cache to store files that may be needed during the period of disconnection. However unlike in Coda, files are not automatically hoarded on the basis of hints from the user. (Support for hoarding seems technically feasible but has not been developed so far.) Instead, the user must "heat up" the cache before disconnecting by actually referencing the files to be carried along. Support for disconnected operation is thus clearly less sophisticated than that provided by Coda. On the other hand, since the modifications to the AFS cache manager require no corresponding changes to the AFS servers, they can be used by the now rather large base of existing AFS users.

Briefly, the AFS cache manager at the client has been modified as follows. Instead of assuming that the cached copy of a file is stale if it does not possess a *callback* [Satyanarayanan 1990], the cache manager takes an optimistic view and allows access to the file. It maintains a log of both mutating and non-mutating AFS operations. Later upon reconnection, it replays these operations to the server. In the case of mutating operations, it compares version-ids and in some cases timestamps to determine if a write-write conflict occurred. Only if it is certain that the file was not modified at the server during disconnection does it actually overwrite the server's copy with the one in its cache. Oth-

erwise, it writes the cached copy to the server using an unused name that it invents, and notifies the user. In the case of non-mutating operations, it tries to detect during the replay if stale data was or may have been used during disconnection. In such cases, it advises the user accordingly.

3.3 Ficus

Ficus [Page 1991] is a distributed UNIX file system developed at UCLA that uses a peer-to-peer model as opposed to the client-server model used by Coda and AFS. It has also been designed to provide high availability of files in the face of failures. To achieve this goal, Ficus replicates files at multiple sites. Both reading and updating of a file is allowed as long as at least one of the replicas is accessible. When an update is made to a replica, Ficus attempts to notify all other replicas of the update. These other replicas then fetch the latest version. Due to failures of replication sites or the network, some replicas may not receive notification of the update. Therefore, replication sites periodically communicate with each other, in an action called *reconciliation*, to find out what updates they have missed and bring their replicas in coherence with each other. Write-write conflicts are guaranteed to be detected. In the case of the directories, such conflicts are resolved by Ficus, while in the case of files, the user is notified.

Ficus is implemented using two cooperating layers: a logical layer that provides the abstraction of a single-copy highly available file and a physical layer that implements the abstraction of a file replica. A single file may be mapped to multiple replicas but the logical layer makes this transparent to the user. The physical layer uses a Unix file system to store a replica locally or NFS to store it remotely. Replication of files is done at the granularity of *volumes*. Volumes are glued together using a mechanism called *grafting* that is similar to Unix mounting. Location information for volumes is stored at the graft points and replication for such information is done in the same manner as for Ficus directories.

Reconciliation of replicas was originally done in a pair-wise manner. However, this proved to be extremely expensive, so reconciliation has been modified to take place in a ring [Heidemann 1992]. If a particular site in the ring is inaccessible during a reconciliation, it is simply skipped during that reconciliation. In order to detect differences between two replicas, timestamps and version vector comparisons are used. Each pair of volume replicas maintains a record of when it started its last rec-

onciliation. Each file replica is also time-stamped with its last update time. If this time is found to be later than the start of the last reconciliation, it may have to be reconciled with another replica this time around. In this case, a comparison of the version vectors of the replicas is conducted to determine if a reconciliation is actually needed.

While support for mobile computing was not a design goal of Ficus, this is one of its useful applications. For example, a user could maintain replicas of critical files from an office workstation on his or her portable, so that they remain available during a trip. Upon returning, the portable is plugged into the network, and reconciliation will automatically bring the replicas on the office machine and the portable in coherence with each other. Ficus can also be used by someone who has machines at home and at office and wishes to keep a portion of their file systems common. While the user commutes, one machine can dial in to the other and updates can be exchanged, so that the latest versions of all files are always available.

In an important respect, Ficus is like Coda: it also uses an optimistic replication strategy to increase availability of files i.e. updates can be made even if only a single replica is accessible and any resulting conflicts are detected and resolved later on. Also, as with Coda, replication is done at the granularity of volumes. Where Ficus and Coda differ markedly is in their models of a distributed system. Coda assumes that a few of the machines in the system have the special status of servers and that the rest are clients. The replicas on the servers are “first-class” replicas and are considered to be of higher quality than the cached copies at clients which are “second-class” replicas. With Ficus, however, all machines and all replicas have equal status.

4. Design

The previous chapters have introduced the broad challenges posed by mobile computing and the work being done to address them with particular attention being paid to work that has been done to provide file system support to mobile computers. In this chapter, we first look at the problems with existing file system support for mobile computing. Next, we present a file system design that uses a PDA to carry the files of most immediate need to its owner. Finally, we look at the relationship between computing personae and file systems and see how the persona concept might be used in designing a file system for mobile users.

4.1 Problem Exposition

A client-server distributed file system such as AFS or Coda provides several things users are looking for in a file system:

- Security - access to files should be under the control of their owners.
- Integrity - files should not be lost due to crashes, disk failures, etc.
- Location transparency - the user sees the same name space at all locations.
- Large disk capacity at a reasonable price.
- The ability to share files with other users in a controlled manner.

The use of trusted and reliable servers is critical to providing each of the above attributes. However, exclusive reliance on them results in limited availability. A solution to this problem is to use the servers of a distributed file system as the “permanent home” for files while permitting the user to temporarily store and access these files elsewhere.

Coda took an important step in this direction through its support for disconnected operation. However, the Coda solution only works if all the computers used by the user are clients of the same distributed file system. There are many situations in which this model does not hold. For example, while at home, the user may use a stationary workstation that has no network connectivity. Coda files cannot be accessed on this workstation even if it can communicate over a short-range link with a portable Coda client carried by the user. Another problem is that Coda does not allow sharing of files between clients that are connected to each other but disconnected from all servers.

Coda maintains first-class replicas at servers and allows second-class replicas only at clients. I propose to extend this model to allow second-class replicas even at a machine that is not a client if the machine is able to communicate with a client. Coda does not allow direct sharing between second-class replication sites that are connected to each other. I propose that such sharing be allowed when no first-class replicas are accessible. Allowing such sharing introduces the problem of maintaining coherency between second-class replicas. This is an interesting and significant problem that still needs to be addressed. Figure 2 graphically depicts the replication model being proposed. The shaded rectangle represents a replica that previously could not exist and the horizontal arrows depict interactions that previously could not occur.

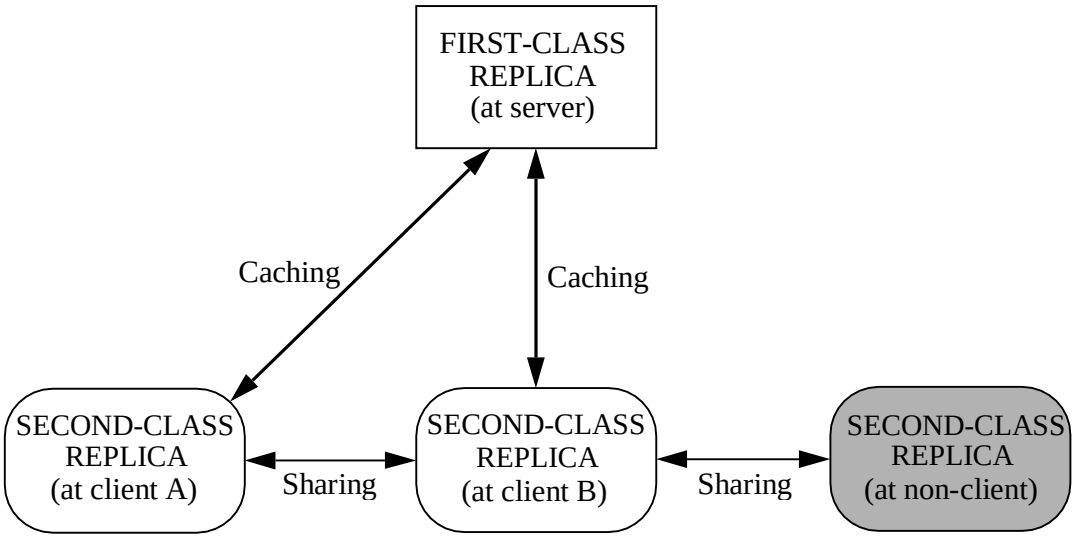


Figure 2: Proposed replication model

4.2 A PDA-based File System Design

The other major deficiency with Coda's support for mobile computing is that it is targeted towards portable workstations and may not work well for PDAs. A typical scenario in which Coda works effectively is as follows: the user brings his or her portable into the office, plugs it into the network, works on it for a while, unplugs it when it is time to leave and then uses it in disconnected mode. Because the user was working on the portable just before disconnection, the active files were automatically loaded into the cache.

Unfortunately, this kind of scenario is unlikely if the portable in question is a PDA. Because

of the PDA's tiny user interface, the tasks for which it is well suited are limited. Therefore, whenever a user has access to a larger computer, he or she is likely to use that one instead. Thus, a much more likely scenario is that while the user is in the office he or she will mostly use the desktop system that is always there. Then, with standard Coda, when he or she leaves the office, the active files would *not* be in the PDA's cache; they would be in the desktop's cache.

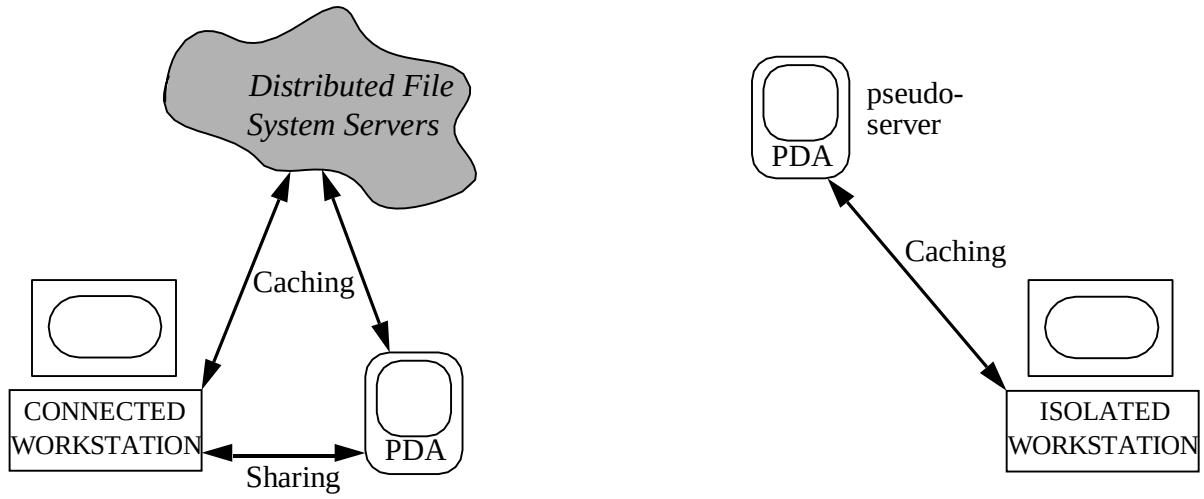
In fact, it seems likely that a PDA, because of its limited user interface, will frequently be used in conjunction with other larger computers that happen to be accessible. Let us consider an example of how it might be used. A user preparing for a business trip starts working on a document using an office desktop system. Through a file sharing mechanism, a copy of the document is concurrently maintained on a PDA. Later, on the cab ride to the airport, the user would employ the PDA to edit the document since there are no other accessible computers. Once on the airplane however, the user might find that the seatback in front has a computer in it that can be used to edit the document resident on the PDA. What is needed to make all this work is a mechanism for transparently sharing files between a PDA and a proximate computer that the PDA's owner wishes to use.

While the PDA's small size is a limitation, it also brings with it the tremendous advantage that it can be carried by its owner at all times. This feature is something we must exploit in designing a file system for a mobile computing environment that includes these devices. Since the PDA is with its owner at all times, it can act as a carrier of currently active or frequently used files.

One solution to the file system needs of a mobile user is a *PDA-based* file system. In such a file system, special status is accorded to the user's PDA because it is always with the user. The permanent home of the user's files is on the servers of a distributed file system such as Coda, and the PDA is used as a link between the distributed file system and the file systems of other independent computers operated by the user.

Whenever circumstances permit, the user works at a connected client of the distributed file system. At such times, the PDA, through a connection with this client or through a direct connection with a server, hoards files that are important to the user (specified using a hoard file) and files that the user is currently working on at the client. This situation is shown in Figure 3(a). When the user has to disconnect from the distributed file system, he or she then depends on the contents of the PDA's cache.

While disconnected, files can be directly accessed on the PDA. However, as shown in Figure 3(b), the PDA can also act as a *pseudo-server* to an isolated workstation (one that is not a client of the distributed file system). The workstation caches from the PDA any files that the user wishes to use. When a file is modified at the workstation or a new file is created there, a copy is sent to the PDA. When the user next connects to the distributed file system, the PDA propagates all changes made while disconnected back to the real server(s).



(a) Operating connected (b) Operating disconnected

Figure 3: A PDA-based file system

An argument against entrusting a PDA with the responsibility of carrying files critical to its owner is that it is very vulnerable to loss, theft or damage. However, we are relying entirely on the PDA only while the user is disconnected from the distributed file system. Therefore, only files that have been created or modified since the last disconnection are vulnerable to loss. This is reasonable risk for the user to take in exchange for increased availability. Also, copies (possibly stale) of lost files may exist on other workstations that were used, thus providing some additional protection.

4.3 Computing Personae and File Systems

A user's computing persona is, roughly speaking, the computing environment that he or she sees. This includes a *name space*, which is generally a name space of files, but which could also be extended (as in the Prospero Directory Service) to include other objects such as printers. It also includes the user's *open files*, *available devices*, the *user interface*, any *communication connections*,

her *active applications* and the *state* of those applications. Some of these components, such as files and network connections, are passive while others, including running applications and communication connections, are active.

Today, when a user makes use of multiple computers, he or she has a computing persona on each one of them. The user has to manually force these multiple personae to resemble each other by copying files, restarting applications, etc. Ideally, the user should be able to move from one location to another and have the persona just follow. There is currently a project under way at the Distributed Computing Research Laboratory to build the various mechanisms needed to allow a computing persona to do just this.

This project is still at an early stage and the details of these various mechanisms are not yet clear. It does seem clear, though, that a persona will require an active entity called a *persona manager*, resident at one or more locations in the system, to monitor the persona and facilitate its movement. When the user logs out from the system at one location, the persona manager "freezes" the persona, which then remains dormant until the user the logs in again (at a possibly different location). Then, the persona manager "unfreezes" the persona and re-establishes it at the user's new location.

The relationship between the file system and the persona manager is two-way. On the one hand, the user's view of the file system is part of the persona and so it is the persona manager's job to manage and move it when needed. On the other, the persona manager needs support from the file system to perform its functions. For example, in order to restart the user's applications at a new location from the same point at which they were left off at the old, it will probably have to store the state of these applications somewhere in the file system.

In a persona-based file system, when the user moves from one location to another, any files created or modified at the old location move along with the user to the new location. If both locations are part of a connected system, this can be handled automatically by a distributed file system such as Coda. Otherwise, the persona manager will have to perform this transfer and is entrusted with the responsibility of holding on to all modified and newly created files. However, as soon as the user moves to a connected client of the distributed file system, these modifications are written to a server,

and the persona manager is relieved of the burden of having to hold on to these files.

It should be noted that a persona-based file system could also be PDA-based. There is no magical way of moving a user's persona from the old location to the new one. There are only two ways for this transfer to occur. If there is a connection (wired, wireless or a combination of the two) between the old and new location, the transfer can use this connection. If such a connection is not available, the only alternative is for the user to physically carry the persona on some device. A PDA is the most logical choice for such a device because of the ease with which it can be carried.

5. Implementation

In the last chapter, we looked at possible solutions to the problem of developing a file system for mobile computing. This chapter presents ideas for proceeding with the actual implementation of these solutions.

For reasons explained in the last chapter, our file system design uses a client-server distributed file system as the "permanent home" for files. Two choices for such a distributed file system come to mind - AFS and Coda. We have chosen to go with Coda for two reasons. One is that its support for disconnected operation is much more sophisticated than what is available for AFS through LITTLE WORK. The other reason is that since it is not a commercial product like AFS, there are much fewer licensing problems. We have been able to obtain both binaries and source code for Coda but currently have neither for AFS. Since our implementation may require modifications to client as well as server code, access to source code is essential. One advantage of working with AFS, though, is that the chances of getting real users to use the file system we develop are much higher. Feedback from actual users is of course invaluable for evaluating our design and refining it.

Currently, we have a single Coda server running on a 486-based IBM PS/2 Model 90 and a single Coda client running on a 386-based Hewlett-Packard Vectra QS/16 (a PC clone). Both machines run the Mach 3.0 operating system. Although at some point our setup will have to include a mobile computer, we currently do not have either a PDA or a portable workstation. Until we do, the implementation will have to simulate mobility by making and breaking connections between machines that are in reality physically connected by a wired network.

The first step is to build a file sharing mechanism between a Coda client and a non-client (one that does not have a connection to a Coda server) where the client acts as a pseudo-server to the non-client. This will require a daemon on the Coda client to service requests from the non-client for Coda files. These requests will have to be serviced using the client's cache since the client will typically be running in disconnected mode at such times. For the non-client, we will have to build a cache manager that will intercept file system calls for Coda files and fetch them from the Coda client if necessary. When a Coda file is closed after writing, the cache manager will write a copy back to the client

machine as well. For the non-client, a machine whose local file system supports the Sun vnode interface will be used since this would allow the cache manager to intercept Coda file system calls. An AIX workstation is a possible candidate.

The next step is to allow sharing between two Coda clients that are connected to each other but disconnected from all servers. Currently a client is only allowed to fetch a file from a server. This will have to be changed to allow the option of fetching files from another client. (This option should be under the control of the user since it is safe only if the user knows the other client to be trustworthy.) Further, the cache manager will have to be modified to propagate updates to the other client. Finally, since a client may update a file that it did not obtain from a server, modifications may be necessary to both the Coda cache manager and the Coda server software to allow such updates to be propagated back to a server upon reconnection.

In a research area such as file systems, it is the actual building of a system that gives one real insight into the problem. Things that appear trivial turn out to be extremely difficult and sometimes even vice versa. The primary objective of the implementation will therefore be to answer the important question of whether the design we have presented is buildable or not. In addition, a successful implementation is a prerequisite for addressing other significant questions we are interested in:

- How useable is the system?
- Does it provide reasonable performance?

Answers to these questions will not only help us to better the implementation but will also allow us to evaluate and improve the design.

REFERENCES

1. *Acharya, Kamal, Attendance Management System Project (April 28, 2024). Available at SSRN: https://ssrn.com/abstract=4810251 or http://dx.doi.org/10.2139/ssrn.4810251*
2. *Acharya, Kamal, Online Food Order System (May 2, 2024). Available at SSRN: https://ssrn.com/abstract=4814732 or http://dx.doi.org/10.2139/ssrn.4814732*
3. *Acharya, Kamal, University management system project. (May 1, 2024). Available at SSRN: https://ssrn.com/abstract=4814103 or http://dx.doi.org/10.2139/ssrn.4814103*
4. *Acharya, Kamal, Online banking management system. (May 1, 2024). Available at SSRN: https://ssrn.com/abstract=4813597 or http://dx.doi.org/10.2139/ssrn.4813597*
5. *Acharya, Kamal, Online Job Portal Management System (May 5, 2024). Available at SSRN: https://ssrn.com/abstract=4817534 or http://dx.doi.org/10.2139/ssrn.4817534*
6. *Acharya, Kamal, Employee leave management system. (May 7, 2024). Available at SSRN: https://ssrn.com/abstract=4819626 or http://dx.doi.org/10.2139/ssrn.4819626*
7. *Acharya, Kamal, Online electricity billing project report. (May 7, 2024). Available at SSRN: https://ssrn.com/abstract=4819630 or http://dx.doi.org/10.2139/ssrn.4819630*
8. *Acharya, Kamal, POLICY MANAGEMENT SYSTEM PROJECT REPORT. (December 10, 2023). Available at SSRN: https://ssrn.com/abstract=4831694 or http://dx.doi.org/10.2139/ssrn.4831694*
9. *Acharya, Kamal, Online job placement system project report. (January 10, 2023). Available at SSRN: https://ssrn.com/abstract=4831638 or http://dx.doi.org/10.2139/ssrn.4831638*
10. *Acharya, Kamal, Software testing for project report. (May 16, 2023). Available at SSRN: https://ssrn.com/abstract=4831028 or http://dx.doi.org/10.2139/ssrn.4831028*
11. *Acharya, Kamal, ONLINE CRIME REPORTING SYSTEM PROJECT. (August 10, 2022). Available at SSRN: https://ssrn.com/abstract=4831015 or http://dx.doi.org/10.2139/ssrn.4831015*
12. *Acharya, Kamal, Burger ordering system project report. (October 10, 2022). Available at SSRN: https://ssrn.com/abstract=4832704 or http://dx.doi.org/10.2139/ssrn.4832704*
13. *Acharya, Kamal, Teachers Record Management System Project Report (December 10, 2023). Available at SSRN: https://ssrn.com/abstract=4833821 or http://dx.doi.org/10.2139/ssrn.4833821*
14. *Acharya, Kamal, Dairy Management System Project Report (December 20, 2020). Available at SSRN: https://ssrn.com/abstract=4835231 or http://dx.doi.org/10.2139/ssrn.4835231*
15. *Acharya, Kamal, Electrical Shop Management System Project (December 10, 2019). Available at SSRN: https://ssrn.com/abstract=4835238 or http://dx.doi.org/10.2139/ssrn.4835238*

16. *Acharya, Kamal, Online book store management system project report. (Febuary 10, 2020). Available at SSRN: https://ssrn.com/abstract=4835277 or http://dx.doi.org/10.2139/ssrn.4835277*
17. *Acharya, Kamal, Paint shop management system project report. (January 10, 2019). Available at SSRN: https://ssrn.com/abstract=4835441 or http://dx.doi.org/10.2139/ssrn.4835441*
18. *Acharya, Kamal, Supermarket billing system project report. (August 10, 2021). Available at SSRN: https://ssrn.com/abstract=4835474 or http://dx.doi.org/10.2139/ssrn.4835474*
19. *Acharya, Kamal, Online taxi booking system project report. (March 10, 2022). Available at SSRN: https://ssrn.com/abstract=4837729 or http://dx.doi.org/10.2139/ssrn.4837729*
20. *Acharya, Kamal, Online car servicing system project report. (March 10, 2023). Available at SSRN: https://ssrn.com/abstract=4837832 or http://dx.doi.org/10.2139/ssrn.4837832*
21. *Acharya, Kamal, School management system project report. (July 10, 2021). Available at SSRN: https://ssrn.com/abstract=4837837 or http://dx.doi.org/10.2139/ssrn.4837837*
22. *Acharya, Kamal, Furniture Showroom Management System Project Report (March 21, 2021). Available at SSRN: https://ssrn.com/abstract=4839422 or http://dx.doi.org/10.2139/ssrn.4839422*
23. *Acharya, Kamal, Online Vehicle Rental System Project Report (March 21, 2019). Available at SSRN: https://ssrn.com/abstract=4839429 or http://dx.doi.org/10.2139/ssrn.4839429*
24. *Acharya, Kamal, Fruit Shop Management System Project Report (August 10, 2023). Available at SSRN: https://ssrn.com/abstract=4841048 or http://dx.doi.org/10.2139/ssrn.4841048*
25. *Acharya, Kamal, Hall Booking Management System Project Report (December 21, 2023). Available at SSRN: https://ssrn.com/abstract=4841055 or http://dx.doi.org/10.2139/ssrn.4841055*
26. *Acharya, Kamal, Lundry Management System Project Report (October 21, 2023). Available at SSRN: https://ssrn.com/abstract=4841059 or http://dx.doi.org/10.2139/ssrn.4841059*
27. *Acharya, Kamal, A CASE STUDY OF CINEMA MANAGEMENT SYSTEM PROJECT (September 25, 2023). Available at SSRN: https://ssrn.com/abstract=4841209 or http://dx.doi.org/10.2139/ssrn.4841209*
28. *Acharya, Kamal, A CASE STUDY ON ONLINE TICKET BOOKING SYSTEM PROJECT (May 25, 2024). Available at SSRN: https://ssrn.com/abstract=4841210 or http://dx.doi.org/10.2139/ssrn.4841210*
29. *Acharya, Kamal, ONLINE DATING MANAGEMENT SYSTEM PROJECT REPORT. (April 25, 2023). Available at SSRN: https://ssrn.com/abstract=4842066 or http://dx.doi.org/10.2139/ssrn.4842066*
30. *Acharya, Kamal, ONLINE RESUME BUILDER MANAGEMENT SYSTEM PROJECT REPORT. (April 25, 2021). Available at SSRN: https://ssrn.com/abstract=4842071 or http://dx.doi.org/10.2139/ssrn.4842071*
31. *Acharya, Kamal, TOLL TEX MANAGEMENT SYSTEM PROJECT REPORT (August 21, 2023). Available at SSRN: https://ssrn.com/abstract=4842082 or http://dx.doi.org/10.2139/ssrn.4842082*

32. *Acharya, Kamal, Chat Application Through Client Server Management System Project Report (June 25, 2023). Available at SSRN: https://ssrn.com/abstract=4842761 or http://dx.doi.org/10.2139/ssrn.4842761*
33. *Acharya, Kamal, Web Chatting Application Management System Project Report (April 25, 2022). Available at SSRN: https://ssrn.com/abstract=4842771 or http://dx.doi.org/10.2139/ssrn.4842771*
34. *Acharya, Kamal, Automobile management system project report (May 25, 2022). Available at SSRN: https://ssrn.com/abstract=4846917 or http://dx.doi.org/10.2139/ssrn.4846917*
35. *Acharya, Kamal, College bus management system project report (April 25, 2023). Available at SSRN: https://ssrn.com/abstract=4846920 or http://dx.doi.org/10.2139/ssrn.4846920*
36. *Acharya, Kamal, Courier management system project report (May 25, 2023). Available at SSRN: https://ssrn.com/abstract=4846922 or http://dx.doi.org/10.2139/ssrn.4846922*
37. *Acharya, Kamal, Event management system project report (April 25, 2021). Available at SSRN: https://ssrn.com/abstract=4846927 or http://dx.doi.org/10.2139/ssrn.4846927*
38. *Acharya, Kamal, Library management system project report II (May 25, 2020). Available at SSRN: https://ssrn.com/abstract=4848857 or http://dx.doi.org/10.2139/ssrn.4848857*
39. *Kamal Acharya. Teacher record management system project report. Authorea. August 02, 2024. DOI: https://doi.org/10.22541/au.172261514.46787329/v1*
40. *Kamal Acharya. POST OFFICE MANAGEMENT SYSTEM PROJECT REPORT. Authorea. August 02, 2024. DOI: https://doi.org/10.22541/au.172261514.44494375/v1*
41. *Kamal Acharya. Fruit shop management system project report. Authorea. August 02, 2024. DOI: https://doi.org/10.22541/au.172261514.42227675/v1*
42. *Kamal Acharya. Dairy management system project report. Authorea. August 02, 2024. DOI: https://doi.org/10.22541/au.172261513.39402347/v1*
43. *Kamal Acharya. DATA COMMUNICATION AND COMPUTER NETWORK MANAGEMENT SYSTEM PROJECT REPORT. Authorea. August 01, 2024. DOI: https://doi.org/10.22541/au.172254873.37480177/v1*
44. *Kamal Acharya. School management system project report. Authorea. August 01, 2024. DOI: https://doi.org/10.22541/au.172254873.34023165/v1*
45. *Kamal Acharya. A CASE STUDY OF CINEMA MANAGEMENT SYSTEM PROJECT. Authorea. August 01, 2024. DOI: https://doi.org/10.22541/au.172254873.30191075/v1*
46. *Kamal Acharya. A CASE STUDY ON ONLINE TICKET BOOKING SYSTEM PROJECT. Authorea. August 01, 2024 DOI: https://doi.org/10.22541/au.172254872.26972790/v1*
47. *Kamal Acharya. Web chatting application project report management system. Authorea. August 01, 2024. DOI: https://doi.org/10.22541/au.172254871.18588592/v1*
48. *Kamal Acharya. RETAIL STORE MANAGEMENT SYSTEM PROJECT REPORT. Authorea. August 01, 2024. DOI: https://doi.org/10.22541/au.172254871.14590154/v1*
49. *Kamal Acharya. SUPERMARKET MANAGEMENT SYSTEM PROJECT REPORT. Authorea. August 01, 2024. DOI: https://doi.org/10.22541/au.172252491.19145062/v1*

50. *Kamal Acharya. SOCIAL MEDIA MANAGEMENT SYSTEM PROJECT REPORT. Authorea. August 01, 2024.*
DOI: https://doi.org/10.22541/au.172252491.11210579/v1
51. *Kamal Acharya. Online music portal management system project report. Authorea. August 01, 2024.*
DOI: https://doi.org/10.22541/au.172252488.89734698/v1
52. *Kamal Acharya. COLLEGE BUS MANAGEMENT SYSTEM PROJECT REPORT. Authorea. July 31, 2024.*
DOI: https://doi.org/10.22541/au.172245277.70798942/v1
53. *Kamal Acharya. AUTOMOBILE MANAGEMENT SYSTEM PROJECT REPORT. Authorea. July 31, 2024.*
DOI: https://doi.org/10.22541/au.172245276.67982593/v1
54. *Kamal Acharya. Ludo management system project report. Authorea. July 31, 2024*
DOI: https://doi.org/10.22541/au.172243999.98091616/v1
55. *Kamal Acharya. Literature online quiz system project report. Authorea. July 31, 2024*
DOI: https://doi.org/10.22541/au.172243825.53562953/v1
56. *Kamal Acharya. Avoid waste management system project. Authorea. July 29, 2024*
DOI: https://doi.org/10.22541/au.172228528.85022205/v1
57. *Kamal Acharya. CHAT APPLICATION THROUGH CLIENT SERVER MANAGEMENT SYSTEM PROJECT. Authorea. July 29, 2024.*
DOI: https://doi.org/10.22541/au.172228527.74316529/v1
58. *Kamal Acharya. Parking allotment system project report. Authorea. July 29, 2024.*
DOI: https://doi.org/10.22541/au.172227078.89966943/v1
59. *Kamal Acharya. HEALTH INSURANCE CLAIM MANAGEMENT SYSTEM. Authorea. July 26, 2024.*
DOI: https://doi.org/10.22541/au.172202020.06707762/v1
60. *Kamal Acharya. ONLINE TRAIN BOOKING SYSTEM PROJECT REPORT. Authorea. July 22, 2024.*
DOI: https://doi.org/10.22541/au.172167914.45160406/v1
61. *Kamal Acharya. COVID MANAGEMENT SYSTEM PROJECT REPORT. Authorea. July 16, 2024.*
DOI: https://doi.org/10.22541/au.172116616.60220024/v1
62. *Kamal Acharya. COVID MANAGEMENT SYSTEM PROJECT REPORT. Authorea. July 16, 2024.*
DOI: https://doi.org/10.22541/au.172116616.60220024/v1